Disks for Data Centers

White paper for FAST 2016

Eric Brewer, Lawrence Ying,
Lawrence Greenfield, Robert Cypher, and Theodore Ts'o
Google, Inc.

February 23, 2016
Version 1.1, revised February 29, 2016

Online at: http://research.google.com/pubs/pub44830.html

Google makes no warranties concerning the information contained in this document.

Abstract

Disks form the central element of Cloud-based storage, whose demand far outpaces the considerable rate of innovation in disks. Exponential growth in demand, already in progress for 15+ years, implies that most future disks will be in data centers and thus part of a large collection of disks. We describe the "collection view" of disks and how it and the focus on tail latency, driven by live services, place new and different requirements on disks. Beyond defining key metrics for data-center disks, we explore a range of new physical design options and changes to firmware that could improve these metrics.

We hope this is the beginning of a new era of "data center" disks and a new broad and open discussion about how to evolve disks for data centers. The ideas presented here provide some guidance and some options, but we believe the best solutions will come from the combined efforts of industry, academia and other large customers.

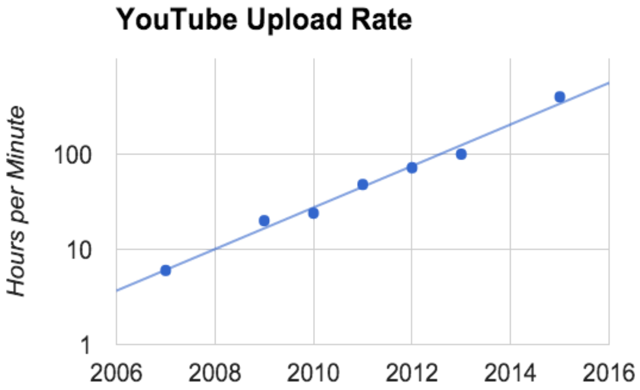
Table of Contents

Acknowledgements: Joe Heinrich, Sean Quinlan, Andrew Fikes, Rob Ewaschuk, Marilee Schultz, Urs Hölzle, Brent Welch, Remzi Arpaci-Dusseau (University of Wisconsin), Garth Gibson (Carnegie Mellon University), Rick Boyle, Kaushal Upadhyaya, Maire Mahony, Danner Stodolsky, and Florentina Popovici

Introduction

The rise of portable devices and services in the Cloud has the consequence that (spinning) hard disks will be deployed primarily as part of large storage services housed in data centers. Such services are already the fastest growing market for disks and will be the majority market in the near future. For example, for YouTube alone, users upload over 400 hours of video every minute, which at one gigabyte per hour requires a petabyte (1M GB) of new storage *every day*. As shown in the graph, this continues to grow exponentially, with a 10x increase every five years.

But the current generation of modern disks, often called "nearline enterprise" disks, are not optimized for this new use case, and instead are designed around the needs of traditional servers. We believe it is time to develop, jointly with the industry and academia, a new line of disks that are specifically designed for large-scale data centers and services.



This paper introduces the concept of "data center" disks and describes the key goals and metrics for these disks. The key differences fall into three broad categories: 1) the "collection view" in which we focus on aggregate properties of a large collection of disks, 2) a focus on tail latency derived from the use of storage for live services, and 3) variations in security requirements that stem from storing others' data. After covering each in more detail, we explore a range of future directions to meet these goals.

The Collection View

The essential difference: disks for data centers are always part of a large collection of disks – designs thus *optimize the collection* with less regard for individual disks. In this sense, current enterprise disks represent a local maximum. The real target is the global maximum using the following five key metrics:

1. Higher I/Os per second (**IOPS**), typically limited by seeks,
2. Higher **capacity**, in GB
3. Lower **tail latency**, covered below,
4. Meet **security** requirements, and
5. Lower total cost of ownership (**TCO**).

At the data center level, the TCO of a disk-based storage solution is dominated by the disk acquisition cost, the disk power cost, the disk connection overhead (disk slot, compute, and network), and the repairs and maintenance overhead.

Achieving durability in practice requires storing valuable data on multiple drives. Within a data center, high availability in the presence of host failures also requires storing data on multiple disks, even if the disks were perfect and failure free. Tolerance to disasters and other rare events requires replication of data to multiple independent locations. Although it is a goal for disks to provide durability, they can at best be only part of the solution and should avoid extensive optimization to avoid losing data.

This is a variation of the "end to end" argument: avoid doing in lower layers what you have to do anyway in upper layers [1]. In particular, since data of value is never just on one disk, the bit error rate (BER) for a *single* disk could actually be orders of magnitude higher (i.e. lose more bits) than the current target of 1 in 10^{15}, assuming that we can trade off that error rate (at a fixed TCO) for something else, such as capacity or better tail latency.

The collection view also implies higher-level maintenance of the bits, including background checksumming to detect latent errors, data rebalancing for more even use of disks (including new disks), as well as data replication and reconstruction. Moderns disks do variations of these internally, which is partially redundant, and a single disk by itself cannot always do them as well. At the same time, the disk contains extensive knowledge about the low-level details, which in general favors new APIs that enable better cooperation between the disk and higher-level systems.

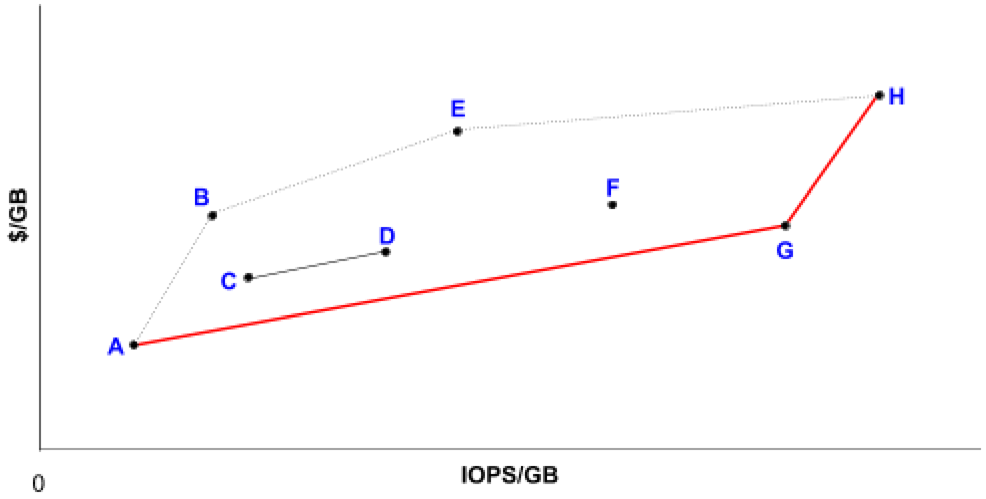
A third aspect of the collection view is that we optimize for the *overall* balance of IOPS and capacity, using a carefully chosen mix of drives that changes over time. We select new disks so that the marginal IOPS and capacity added bring us closer to our overall goals for the collection. Workload changes, such as better use of SSDs or RAM, can shift the aggregate targets.

An obvious question is why are we talking about spinning disks at all, rather than SSDs, which have higher IOPS and are the "future" of storage. The root reason is that the cost per GB remains too high, and more importantly that the growth *rates* in capacity/$ between disks and SSDs are relatively close (at least for SSDs that have sufficient numbers of program-erase cycles to use in data centers), so that cost will not change enough in the coming decade. We do make extensive use of SSDs, but primarily for high-performance workloads and caching, and this helps disks by shifting seeks to SSDs. However, SSDs are really outside the scope of this paper, which is about disks and their role in very large scale storage systems.

The following graph shows one way to think about this: the letters represent various models of disks in terms of their $/GB (in TCO) versus their IOPS per GB.[1] The ideal drive would be *down* and to the right. Newer drives of the same form factor tend to be lower (less $/GB), but towards the left (since they get bigger *more* than they get faster). Changing to smaller platter size is one way to move toward the right. The aggregate properties of a mix of any two drives form a line

[1] The letters do not represent actual disk models; the graph represents how we think about the space. An earlier version of this graph had the y-axis as GB/$ (rather than $/GB), which is more intuitive, but the convex hull property does not hold in that case. This updated version is what we use internally.

between the points, for example the line between C and D. In general, such lines must go up and to the right; if they go down and to the right, the drive on the right is strictly superior and there is no need to mix them (e.g. F dominates E). The optimal mix for a certain IOPS/GB target can be met by combining drives along the bottom of the convex hull, shown as a set of red lines.



This works well as long as we can get most of *both* the capacity and IOPS out of each drive, which is not easy. The industry is relatively good at improving $/GB, but less so at IOPS/GB, and this distinction affects most of the suggested directions below.

A related consequence is that we do not care about precise capacity, e.g. exactly 6TB, or even that the usable capacity of a drive be constant over time. For example, the loss of a head should not prevent the use of the rest of the capacity and IOPS.

Services and Tail Latency

Data center disks require better control over **tail latency**, due to the focus on serving data for remote users while they wait. Broadly speaking, we distinguish between *live services*, in which a user is waiting for a response and low tail latency is critical, and *throughput* workloads, such as MapReduce or Cloud Dataflow, in which performance is throughput oriented and the operations are lower priority. We can often hide the latency of disk writes, so we focus on the tail latency of reads, which we cannot hide in general (and which caching only improves slightly). Our primary metric is thus some form of tail latency for reads, such as the time for the 99th-percentile read; i.e., 99% of reads are within this time bound.

As with the collection view, the focus on tail latency has a range of consequences:

- We label our reads as “low latency” or “throughput” so that we can optimize them appropriately throughout the stack, and so we can monitor them independently. We also have other background “best effort” workloads as well.
- We do admission control on low-latency reads by having a quota for them; otherwise, all reads would have to be faster than average. Over-quota reads are not promised low latency.

- For better overall tail latency, we sometimes issue the same read multiple times to different disks and use the first one to return. This consumes extra resources, but is still sometimes worthwhile. Once we get a returned value, we sometimes *cancel* the other outstanding requests (to reduce the wasted work). This is pretty likely if one disk (or system) has it cached, but the others have queued up real reads. (See Dean and Barroso [2] for more information on tail latency and cancellation.)

Security

Although most of this paper is about changes in prioritization due to a new use case, security requirements fall in the “must have” category, at least over time.

The first general problem is that the size and complexity of the firmware inside a modern disk lead to bugs, including security bugs that can be used to attack the disk or even the host. Hard-disk firmware attacks are not only possible, but appear to have been used [3,4]. Solving this is beyond the scope of this paper, but it is clear that it must be easier to assure correct firmware and restrict unauthorized changes, and in the long-term we must apply the full range of hardening techniques already used in other systems. We approach this problem in the short term by restricting physical access to the disks and by isolation of untrusted code from the host OS (which has the power to reflash the disk firmware).

The second broad category is the need to ensure the data is “encrypted at rest” (i.e. the data on a platter is encrypted in case a device is stolen or incorrectly reused), and access to different types of data are controlled through strict authentication. All modern enterprise disks are already encrypted at rest today, but traditionally with a single key. A side effect of the collection view is that we mix a wide variety of types of data from different customers and with a range of restrictions on the same physical disks. Fine-grained access control, using different keys for different areas of the disk, allows us to use different keys for different customers or restrictions, and in general better implements the principle of least privilege.

The Trusted Computing Group (TCG) has developed some good solutions (TCG OPAL [5] and Enterprise [6]), but the adoption of fine-grained access control is still in its infancy. Further, the use of TCG, especially in SATA drives, has been quite limited and hence there is limited enthusiasm so far for investing in this area. Our hope is that fine-grain access control will become widely deployed, well implemented and tested, and viewed as a standard aspect of best practices.

We will not dig deeper into security, but instead explore a variety of design directions in more detail, broadly grouped into physical changes to disks and firmware changes. For each topic, we label which of the four key metrics it helps to address (TCO, tail latency, IOPS, capacity).

Physical Design Options

Standards last a very long time and thus should be developed thoughtfully and with broad participation. These options are thus the beginning of a long conversation with many constituents. But the shift in usage is real and the potential of new designs increases every year, and thus some new design will make sense in the near future. These options are but one set of proposals, and we know the industry and others will contribute many more.

1) Alternative Form Factors [TCO, IOPS, capacity]

The current 3.5" HDD geometry was adopted for historic reasons – its size inherited from the PC floppy disk. An alternative form factor should yield a better TCO overall. Changing the form factor is a long-term process that requires a broad discussion, but we believe it should be considered. Although we could spec our own form factor (with high volume), the underlying issues extend beyond Google, and developing new solutions together will better serve the whole industry, especially once standardized.

In terms of the platter size, we question whether 3.5" or 2.5" form factors are optimal points. A larger platter size increases GB/$, but lowers IOPS/GB, while smaller platter size results in the opposite. A similar argument applies to the common rotational speed of 7200 RPM, where higher RPM (rotations per minute) increases IOPS/GB, but lowers GB/$, while a lower RPM results in the opposite.

We propose increasing the allowable height ("Z height"). Current disks have a relatively small fixed height: typically 1" for 3.5" disks and 15mm maximum for 2.5" drives. Taller drives allow for more platters per disk, which adds capacity, and amortizes the costs of packaging, the printed-circuit board, and the drive motor/actuator. *Given a fixed total capacity per disk,* smaller platters can yield smaller seek distances and higher RPM (due to platter stability), and thus higher IOPS, but worse GB/$. The net result is a higher GB/$ for any specific IOPS/GB that could be achieved by altering any other single aspect, such as platter sizes or RPM alone. It may also be that a mix of different platter sizes (in different disks) provides the best aggregate solution.

There are a range of secondary optimizations as well, some of which may be significant. These include system-level thermal optimization, system-level vibration optimization, automation and robotics handling optimization, system-level helium[2] optimization, and system-level weight optimizations.

Parallel Accesses [IOPS]

There are multiple ways to enable parallel accesses of two or more IO streams simultaneously, and some implementations benefit a particular workload more than another. Companies have mostly avoided these designs in the past due to their extra cost, but as capacity continues to

[2] Some disks use helium inside the drive to reduce seek time (due to its low density), but it requires sealing the platter area, which would be easier with different dimensions.

outpace growth in IOPS, these options become more compelling over time. Here are four options that could make sense in the future:

1) Two standard-sized actuator arm units, mounted diagonally from each other across the disk platter, each with heads that cover all platter surfaces. This implementation is the most expensive, but can benefit all workloads. Note that this has been implemented before [7,8], although perhaps before its time.
2) One actuator location, but two half-height actuator arm units, mounted on top of each other, each with heads that cover half of the platter surfaces. This implementation can benefit both multi-queue random access or single-queue sequential workloads.
3) One actuator arm unit, with a small dual-stage actuator design that allows two heads to track two platter surfaces simultaneously. This implementation can double sequential workload throughput, but does not help with random access.
4) Once actuator arm with heads that can read two adjacent tracks per platter surface simultaneously. This implementation can double sequential workload throughput.

Although both sequential throughput and random access rates are important, random accesses are particularly important in the disk collection model described here as many users share each drive and some accesses have strict tail-latency bounds. Furthermore, areal density improvements have already resulted in more improvement in sequential throughput than in random-access performance.

Multi-Disk Packages [TCO]

Given we buy so many disks, it would make sense to buy groups of disks as one unit. Many vendors sell disk systems, such as NAS devices, but such systems are too high level and do not provide enough control. Instead we would group disks in order to 1) share larger caches (covered below), 2) amortize fixed costs, and 3) improve power distribution (covered next). The larger package might also help with vibration and yield (see “Flexible Capacity” below). The package could still use one or more SATA interfaces, or could be changed to PCI-E (see “Caching” below).

It is not clear yet how many disks should be in a group. Larger groups have better amortization and power savings, but also take out more data in a failure, which implies higher recovery costs. A group typically also has a higher failure rate than its constituent disks, assuming a failure of one disk requires replacing the group. Four disks might be a reasonable place to start.

Power Delivery [TCO]

The current power delivery from the server to the disk is typically via a connector that delivers some of 12V, 5V and 3.3V (all DC). Each voltage has a current spec that the server must meet, and overall there is considerable mismatch between what is needed to support a generic disk (all voltages at full current) and what is needed for any specific model.

Instead, we propose a single 12V DC supply with a max current (or alternative voltages[3] such as 48V). This can be useful in some modern designs where the overall storage system does not even use 5V or 3.3V anywhere else.

2) Cache Memory [TCO, tail latency]

Currently, we connect many disks to the same host. Each disk has its own read and write caches of considerable size (30-100MB). From a TCO perspective it makes more sense to move RAM caching from the disks to the host or tray, as a single big cache will be both cheaper and more effective. It is more effective in part due to being larger and shared, but also due to the fact that we can more easily improve the caching policy over time.

That being said, based on traces we know that the cache hit rates in the disks are relatively high, despite the fact that we already have caching in multiple higher layers. This is surprising and still needs more exploration, but we suspect the main reason for this is the effectiveness of read-ahead (and read-behind). If so, most of the RAM actually in the disk should be used for this purpose, in addition to write buffering. Furthermore, the disk can do read-ahead and read-behind when it's free – even when there are pending requests, since those requests might require some rotational time to service. The host does not have visibility into when it is free to extend a read.

One possibility longer term is to keep the same interface and the same fixed amount of RAM per disk, but have the disks use host memory via PCI-E, without any other changes to the disk caching algorithms. Host memory is cheaper per MB, so this should be a cost savings. Host memory is also farther away, so perhaps this would be a performance problem. This is a pretty big change and would only make sense in the context of moving to PCI-E, perhaps as part of supporting multi-disk packages, where the extra bandwidth matters and the cost is easier to amortize.

We can try to combine the disk's ability to read data without performance cost and the host's ability to better manage the lifetime of speculative reads by having the host give reads with flexible bounds: "I want at least 12 KB starting at this LBA, and I'm willing to accept up to 900 KB if it doesn't delay servicing another I/O." This API change for variable reads is similar in spirit to "range writes" in which the disk has multiple options and picks one based on its extra internal knowledge [10].

3) Optimized SMR Implementation [capacity]

Shingled magnetic recording (SMR) allows a hard disk to achieve a higher areal density – typically 10-20% today, with potentially higher capacity gains in the future – by limiting the ability to perform random writes on the physical media. In particular writes must be done in order, like

[3] Google, Intel, and others presented "Future of Power Efficiency and Technology for Green Computing" during DesignCon 2016 to advocate for 48V server designs [9].

shingles, and the writes destroy the next track. As discussed, the value of a disk comes not just from its capacity, but also the number of useful I/O requests it can service. Because of the write restrictions imposed by SMR, when data is deleted, that deleted capacity can not be reused until the system copies the remaining live data in that SMR zone to another part of the disk, a form of garbage collection (GC).

Some data has lifetimes that are more predictable, which allows the storage system using a host-managed SMR drive to reduce the GC overhead by grouping data with the same predicted lifetime into an SMR zone. Without this, the GC overhead in terms of both the IOPS and capacity becomes significant. Storing only “SMR friendly” data on the SMR disks could mitigate the problem, but this data is generally colder than the rest of the data, which means some of the IOPS on the SMR disks will be wasted. Worse, removing the SMR-friendly data concentrates the IOPS requirements for the remaining hotter data on the conventional spindles (“conventional magnetic recording” or CMR), which increases the IOPS needed on the CMR disks. More broadly, SMR drives reveal a bias towards GB/$ improvements over IOPS/GB improvements.

One way of addressing this is to mix CMR and SMR technologies in a single, hybrid hard drive. This better mixes hot and cold data on the same disk and increases the chance of effectively using both the capacity and the IOPS. In particular, this allows new data to be stored in CMR, with SMR space used for 1) new data known to be long-lived or to have a predictable lifetime (when initially written), or 2) older data stored in CMR that has aged to the point where it is presumed to be long-lived and can be moved to SMR.

The simplest implementation of a hybrid drive is to use a mix of platters and heads, with some optimized for SMR, and the rest for CMR. This gives a permanent fixed ratio of CMR to SMR, which may become suboptimal over time. Alternatively, if a CMR optimized head is used for both SMR and CMR recording, the outer tracks could be used for CMR, while the inner tracks could be used for SMR. Note that the SMR capacity gain will be less in this case, but the CMR portion of the disk will see IOPS improvements due to the lower average seek distance.

Another way to reduce GC overhead is by relaxing the write restrictions imposed by defining a host-managed SMR abstraction. An example is “Caveat Scriptor” [11], which allows the host to write randomly, knowing that it will destroy (specific) nearby data. The Caveat Scriptor proposal also supports other more interesting write patterns, including a circular buffer, and effectively allows the host to have dynamically resizable SMR zones. In the long term, an even more aggressive approach would be to allow dynamically configurable CMR and SMR tracks, which may be achievable with more advanced head-media interface (HMI) technologies.

Firmware Directions

Although the long term should include physical design changes, there are a wide range of firmware-only changes that can be made in the short term using existing form factors. We thus expect these to the primary focus over the next several years, hopefully in parallel with longer-term discussions about new physical designs.

4) Profiling Data [IOPS, tail latency]

An important goal, both short term and long term, is to enable us to understand what is going on with the disk in terms of performance. This data would improve both the host software and the design of disk trays.

One approach would be a “profile data log” implemented as either counters or a ring-buffer in RAM. This data can be read using either a new command or normal reads of magic block numbers. The contents of the log would be important performance or other firmware background events, such as rewrites for data integrity. The host would periodically read this log, without any performance penalty from the disk (other than using some bandwidth). For example, when the host noticed a high-latency read, it could then read the log for diagnosis.

A related idea would be to increase the number of possible error codes returned by a read to enable fine-grain explanations of what caused a delay. In most cases, such reads return correct data but too slowly, and the error code reveals why.

At a minimum, we envision profiling returning the following information:

- time spent seeking (including rotational delay) for a host-initiated read,
- time spent transferring data for a host-initiated read,
- time spent in error recovery for a host-initiated read.

Likewise, profile data should include counters for host-initiated writes and for drive-initiated reads and writes. We also want drives to accumulate separately the time for major background activities (e.g. background media scans, track rewrites).

5) Host-Managed Read Retries [tail latency]

The goal is to allow for different policies for reads. High-end disks allow some management of reads at a global level, but the goal here is to be able to change the read policy on each read, which requires an API change. The obvious default policy would be the current one, “really try hard”, in which the disk goes to great lengths to complete the read. However, most reads would be issued with a “limited retry” policy, which returns an error to the host (quickly) if a few initial attempts at reading returns bad data. We prefer this approach for tail latency and it fits well with the parallel reads and cancellations mentioned above (and in more detail below). In addition, after a “limited retry” read fails, the host still has the option to use the “really try hard” read for the same data.

6) Target Error Rate [capacity, tail latency]

From the view of a single server, it is very important that disk drives do not lose data. Disk drives thus promise incredibly low loss rates, typically losing 1 bit in 10^{15} or less. However, in the data center context, durable data must already be spread across multiple disks and typically multiple locations, so that disasters are unlikely to destroy all copies of the data. High availability also implies the need for multiple copies.

At the same time, current drives make many sacrifices to achieve these low error rates. They sacrifice capacity by limiting how tight the areal density can be and by using more extensive coding, and they worsen tail latency, by requiring more background repair processes that interfere with the normal performance of the disk.

Overall, a higher target error rate would have little effect on the overall durability, while at the same time enabling higher capacity and fewer repair operations. It might be possible to eliminate repair operations *inside* the disk altogether, although the drive still must enable detection of errors (and possibly predict risk of errors) so that higher-level systems can initiate repair and/or shift responsibility for that data to another drive.

7) Background Tasks and Background Management APIs [tail latency]

The disk uses background tasks to fix or prevent problems. One example is refreshing tracks for data integrity, more commonly known as ATI mitigation (for “adjacent track interference”). Background tasks damage tail latency severely, primarily because they can impact important latency-sensitive reads.

Since we cannot remove the need for background tasks altogether, we need to explore mechanisms to better control the timing. Our best suggestion so far is the following contract:

a) Most background tasks should be preemptible.
b) There are no non-preemptible background tasks without a timer. The tasks can potentially be grouped into “urgent” (needs to be executed relatively quickly) vs. “non-urgent” (can be executed less quickly).
c) The host will periodically issue a command that suggests when the disk drive can do background work. This could be a repair for a single track, or for “up to k” tracks when we believe there is more time. The latter allows for better seek ordering.
d) The disk will have an estimate and reporting mechanism for how much background work is piling up.
e) When the timer expires, the disk can also do maintenance, regardless of tail latency. The host should not allow this to happen in general. It might also make sense to have the disk enter maintenance mode any time the refresh count is above some high-water mark (and stop when the count gets below some low-water mark). It would still process commands, but with no promises on tail latency. The host should have a way to tell the disk is in this mode.
f) If new commands are not arriving, something may be wrong with the host and thus doing all the maintenance work now is a good idea. This implies some kind of idle timer that initiates maintenance mode. The disk stays in that mode until a command arrives.

There are some problems with this as is that merit discussion. First, it may be that a bad pattern, such as overwriting the same track many times in a row, may require background work that does not happen in time. This will result in the background task being escalated from the “non-urgent” to the “urgent” bucket. If the host does not allow the disk to initiate the background task while in the “urgent” bucket for too long, then data may potentially be lost or corrupted.

One option would be to allow the data loss, which might be fine depending on the frequency. When the refresh does finally occur, it should be clear whether data was lost or not, and we can return an error at that time.

Another option would be to fail a new write that is *about* to cause excess interference with a nearby track, with an error that means “urgent refresh required” (for that area of the disk). Completing the write a little later is fine if rare. Plus, this kind of error should itself have low latency. Even better might be to do the refresh instead of the write (and return an error), or do the refresh *and* the write, since they are physically close. The latter is bad only if the latency is too high, since we are starving reads during the double write. An advantage of this approach is that it handles pathological cases (better than timers), such as the host repeatedly writing the same track. The best approach here remains an open question.

Background scanning [tail latency]

Many disks do media scanning across all sectors, whether or not they’re allocated, as a way to detect latent errors. Alternatively, we can do background scanning on the host side, but with a more end-to-end approach. However, the primary source of latent errors is the disk itself, due to physically nearby writes in the form of ATI (adjacent track interference) or FTI (far track interference), and thus the disk has a better estimate of which tracks or sectors are most at risk.

One approach is to use a mix of host- and disk-initiated background scans, assuming we can control the timing as discussed above. In particular, we could make the disk responsible for *estimation* of risk and then report to the host the tracks in need of refresh writes. This leaves the scheduling in the hands of the host, which provides better control over the timing and thus helps tail latency. Similarly, if the disk detects a partial (or complete) error it could notify the host, even if the errors were corrected via coding. Such notifications reduce the window of vulnerability to losing this copy; i.e., higher-level repairs can be made before there is a problem.

For LBA ranges that are not specifically called out by the disk as being at risk, the host could promise to issue a special read command (or overwrite) to every allocated sector every X days (and have some way of knowing what the manufacturer thinks is the right value of X).

For LBA ranges that are not scanned by the host, the disk should continue to ensure that they are protected using its own BER guarantee scheme and whatever scanning that implies. Auditing for host-initiated scans can be done and reported, which would be useful, as long as the number of distinct LBA ranges (or bands) is relatively small.

8) Flexible Capacity [TCO, capacity]

Currently, drives come in fixed capacities (such as “X TB”). There are three ways in which moving to flexible capacity should reduce TCO.

First, in order to provide marketable drives with fixed numbers of TBs of capacity, vendors must provision platters with somewhat larger capacities in order to accommodate an unknown number of defects. Drives that have average or below average numbers of defects will therefore be re-manufactured, or sold as having a smaller capacity than they can actually support.

Second, drives have numerous reserved sectors for reallocation, cache, or other uses. Some of those reserved sectors could be exposed to store data early in the life of the drive, provided that the drive is able to reclaim them at a slow rate during its lifetime.

Third, drive head failures could be handled by mapping out the failed head (and recovering the lost data from other drives), although this likely require reinitializing the drive to achieve contiguous LBA numbering. Continuing to use the drive after one or more head failures would extend the lifetime of the drive.

9) Larger Sector Sizes [capacity]

Disk drive error-correcting codes (ECCs) are able to tolerate a given fraction of errors with a smaller fraction of check bits as the codeword size is increased. However, the codeword cannot exceed the drive’s sector size (unless multiple physical sectors are read per logical sector read). As a result, drive vendors have increased the drive sector size from 512B to 4KiB and reduced the ECC overhead. Since storage services typically write their own distributed file systems, there may be an advantage to further increasing this to 64KiB or larger.

Furthermore, since host software typically adds CRC to data written on disks, having SATA drives that support “extended” sector sizes such as 4k+16B or 64k+256B would be more efficient overall, and would allow the ECC to be exposed to the host for end-to-end checksumming. (Some SCSI disks already support this feature.)

10) Optimized Queuing Management [IOPS]

Native command queueing (NCQ) allows IOs to be queued at the disk instead of the host, enabling higher throughput to be achieved through real-time geometry optimizations. Currently, NCQ by itself does not satisfy our need (nor does TCQ), as it does not convey the host’s per I/O requirements to the disk to allow optimal I/O reordering. There are a number of traffic classes mixed with a quota management system in a shared distributed file system. Any attempt to increase throughput by reordering requests has to address two key challenges.

First, it is necessary to meet the latency targets for in-quota low-latency and throughput reads. Second, it is necessary to meet per-user throughput targets for in-quota reads and writes.

Given the state of NCQ and its expected evolution in the future, it seems natural to have the host manage the quota system through throttling, and have the disk manage the per traffic class throughput through informed I/O reordering. A much more detailed investigation and a well thought out design is needed in this area, which is beyond the scope of this document. The disk scheduler will need to look very much like a real-time operating system (RTOS), capable of handling very complex per-I/O information, matching that against real-time head and media-relative positioning, and allowing scheduling interrupts to happen instantaneously as they arise. The disk scheduler will also need to support the queueing and proper scheduling of system management commands, such as periodic environmental monitoring commands.

Some potential APIs needed for the host in this area might include: fine-grained per-I/O priority level information, fine-grained per-I/O deadline information, per-I/O lightweight cancellation, queue barrier insertion, head-of-queue insertion, head-of-priority-group insertion, and general queue management information (such as pulling the detailed state of the current queue and expected execution ordering). Some of the ideas in NCQ ICC can potentially be used for this purpose, but NCQ ICC itself is often considered to be too complex to be fully implemented at the HDD firmware level, and is not fundamentally designed to support the needs of a distributed, data-center-scale file system.

A related use would be to use NCQ commands to issue a giant streaming read. The idea is that if we know we want to stream many megabytes over the next k milliseconds, we would issue a series of low-priority reads, with the more near-term reads having higher priority. The goal is a simple way to interleave blocks for the stream with regular reads/writes. It would also be good to have cache controls for these blocks: once a (streaming) read is delivered to the host, it can be removed from the cache, as there is no expected temporal locality.

Summary

Disks are the central element of Cloud-based storage, whose demand far outpaces the considerable rate of innovation in disks. Exponential growth in demand implies that most future disks will be in data centers and thus part of a large collection of disks. The collection view, along with a focus on tail latency and security, place new and different requirements on disks.

We hope this is the beginning of a new era of “data center” disks and a new broad and open discussion about how to evolve disks for data centers. The ideas presented here provide some guidance and some options, but we believe the best solutions will come from the combined efforts of industry, academia and other large customers.

Bibliography

[1] J. H. Saltzer, D. P. Reed and D. D. Clark. "End-to-End Arguments in System Design". In: Proceedings of the Second International Conference on Distributed Computing Systems. April 1981. IEEE Computer Society. See Wikipedia: https://en.wikipedia.org/wiki/End-to-end_principle

[2] J. Dean and L. Barroso. “The Tail at Scale” *Communications of the ACM*, Vol. 56, No. 2, pp.74-80, February 2013.

[3] S. Malenkovich. “Indestructible malware by Equation cyberspies is out there – but don’t panic (yet)”, Kapersky Blog. February 17, 2015. https://blog.kaspersky.com/equation-hdd-malware/7623/

[4] K. Zetter. “How the NSA’s Firmware Hacking Works and Why It’s So Unsettling.” *Wired* online, February 2015, http://www.wired.com/2015/02/nsa-firmware-hacking/

[5] Trusted Computing Group. “TCG Storage Security Subsystem Class: Opal” Specification Version 2.00, Revision 1.00, February 24, 2012 https://www.trustedcomputinggroup.org/files/resource_files/B15F1F8F-1A4B-B294-D03F09D5122B21F6/Opal_SSC_2%2000_rev1%2000_final.pdf

[6] Trusted Computing Group. “TCG Storage Security Subsystem Class: Enterprise” Specification Version 1.0, Revision 1.0, January 27, 2009 http://www.trustedcomputinggroup.org/files/resource_files/87FE6847-1D09-3519-ADF6E65680700A9F/TCG_SWG_SSC_Enterprise-v1r1-090120.pdf Also presentation: https://www.trustedcomputinggroup.org/files/resource_files/0B968DDB-1A4B-B294-D02FF4F402F72707/SWG_TCG_Enterprise%20Introduction_Sept2010.pdf

[7] C. Kozierok. “Single vs. Multiple Actuators” in *The PC Guide* (http://www.PCGuide.com). April 2001. http://www.pcguide.com/ref/hdd/op/actMultiple-c.html

[8] Wikipedia. “Conner Peripherals”, see “Chinook dual-actuator drive”, https://en.wikipedia.org/wiki/Conner_Peripherals#Performance_issues_and_the_.22Chinook.22_dual-actuator_drive

[9] R. Merritt. “Google, Intel Prep 48V Servers.” *EE Times*, January 21, 2016. http://www.eetimes.com/document.asp?doc_id=1328741

[10] A. Anand, S. Sen, A. Krioukov, F. Popovici, A. Akella, A. Arpaci-Dusseau, R. Arpaci-Dusseau, S. Banerjee. “Avoiding File System Micromanagement with Range Writes” Proceedings of the 8th USENIX conference no Operating Systems Design and Implementation (OSDI 2008). Pp. 161-176.

[11] T. Feldman and G. Gibson. “Shingled Magnetic Recording: Areal Density Increase Requires New Data Management.” *Usenix ;login:*, Vol. 30, No. 3, June 2013. https://www.cs.cmu.edu/~garth/papers/05_feldman_022-030_final.pdf